Konsep *Collection*

IF2210 – Semester II 2022/2023

SAR

Collection

- Di dunia nyata maupun dalam memrogram, seringkali kita berurusan dengan sekumpulan benda yang sejenis.
- Contoh:
  - Lapangan parkir berisi sekumpulan mobil
  - Sekumpulan barang yang dibeli dari supermarket di dalam kantong belanja
  - Menu di resto merupakan sekumpulan *tuple* ⟨nama makanan, harga⟩
  - Daftar semua mahasiswa yang mengambil mata kuliah IF2210
  - Deretan kursi di ruang bioskop
  - dll.
- Dalam konteks pemrograman (khususnya OOP) kumpulan tersebut disebut dengan *collection*.

Jenis-jenis collection

- Setiap *collection* memiliki karakteristik tertentu tergantung kegunaannya:
  - ***Ordered collection*** memiliki suatu keterurutan
    - Misalnya list/array yang keterurutannya ditentukan oleh indeks, e.g, ada elemen ke-0, elemen ke-1, dst
  - ***Sorted collection*** seperti *ordered collection*, dengan keterurutan yang merupakan hasil suatu kalkulasi, i.e., terurut berdasarkan value
  - ***Set*** elemennya tidak boleh berulang
  - ***Dictionary/Map*** merupakan sekumpulan *value* yang diakses melalui *key*
    - ***Bag***, *collection* yang berisi daftar item dan berapa kali item tersebut muncul, seperti kantong belanja yang (misal) berisi 12 telur, 2 susu, dan 3 kentang ⟶ map<item,int>
  - ***String*** dapat dipandang sebagai sebuah *ordered collection of chars*
  - Terkadang dibutuhkan *collection* yang jumlah elemennya tetap (misal: pemain basket dalam satu tim yang sedang bermain), terkadang jumlah elemen harus fleksibel
  - dll.

Apa yang bisa dilakukan dengan collection? (1)

- **Akses:** kapasitas & jumlah elemen yg terisi (umum), mengakses sebuah elemen dalam *collection* (tergantung jenis *collection*)
- **Enumerasi:**
  - **For each:** melakukan hal yang sama kepada setiap elemen
  - **Select/filter:** menghasilkan sub-collection berdasarkan suatu kriteria (misal: mahasiswa yang IPK-nya > 3)
    - **Find/detect:** ambil satu saja dari hasil select
  - **Collect/map:** menghasilkan *collection* baru dengan jumlah elemen yang sama, setiap elemen yang baru merupakan hasil transformasi/operasi terhadap elemen yang lama (contoh: perkalian skalar dengan vektor adalah sebuah collect/map, di mana 3 * [1, 2, 3] = [3*1, 3*2, 3*3])
  - **Reduce, fold:** menghasilkan satu nilai dari semua elemen (misal: sum, average, max)

Apa yang bisa dilakukan dengan collection? (2)

- **Testing:** memeriksa apakah elemen-elemen *collection* memenuhi suatu kondisi tertentu
  - Is empty, is full
  - Any: apakah **ada** elemen yang memenuhi suatu kondisi
  - All: apakah **semua** elemen memenuhi suatu kondisi
- **Menambah & mengurangi elemen:**
  - Menambah & mengurangi 1 elemen
  - Menambah semua elemen dari sebuah *collection* ke *collection* lain
  - Mengurangi elemen yang memenuhi kriteria
  - *Collection* dapat memiliki batasan spesifik dalam penambahan & pengurangan elemen, e.g., Queue dapat dipandang sebagai *collection* (sekumpulan objek) yang hanya bisa ditambah di belakang dan dikurangi di depan

Collection dalam OOP

- Bahasa OO biasanya menyediakan kelas-kelas untuk menangani *collection*, baik *built-in* maupun dalam bentuk *library* tambahan
  - Java: Collection API or Collection Framework (JCF), C++: STL, dst.
- Dalam pemrograman prosedural, aksi terhadap *collection* (seperti pada slide sebelumnya) dilakukan dengan iterasi/*loop* terhadap semua elemen
  - jika dilakukan di OOP, itu artinya meng-*expose* isi dan cara kerja internal *collection* ⟶ menyalahi enkapsulasi
- Pada OOP, *collection* adalah objek juga, bisa dikirimi *message* seperti contoh pada slide berikutnya
- Iterasi/*loop* seharusnya hanya boleh terjadi **di dalam** kelas-kelas *collection*; kelas yang memanfaatkan *collection* **hanya menggunakan** *method* yang di-*expose* kelas-kelas *collection*
  - Terkadang tidak dimungkinkan karena keterbatasan desain bahasa pemrograman yang digunakan

Contoh (1) dalam *Java-like syntax*

- **For-each:** cetak semua isi `array`:
  - Prosedural:
    ```
    for (i=0; i<10; i++) print(array[i]);
    ```
  - OOP:
    ```
    array.forEach(elmt -> print(elmt));
    ```
- **Select/filter:** isi `arr2` dengan elemen `arr1` yang genap saja:
  - Prosedural:
    ```
    j = 0;
    for (i=0; i<10; i++)
        if (arr1[i]%2 == 0)
            arr2[j++] = arr1[i];
    ```
  - OOP:
    ```
    arr2 = arr1.select(elmt -> elmt%2 == 0);
    ```

Perhatikan dengan OO untuk banyak hal jadi tidak perlu peduli jumlah elemen.

Contoh (2) dalam *Java-like syntax*

- **Collect/map:** perkalian skalar dengan vektor, $\boldsymbol{v}_2 = 3 \cdot \boldsymbol{v}_1$:
  - Prosedural:

```
for (i=0; i<3; i++)
    v2[i] = 3 * v1[i];
```

  - OOP:

```
v2 = v1.map(component -> 3*component);
```

- **Testing:** "Boleh masuk jika ada anggota rombongan yang dewasa":
  - Prosedural:

```
bolehMasuk = false;
for (i=0; i<10; i++)
    if (rombongan[i].usia > 17)
        { bolehMasuk = true; break; }
```

  - OOP:

```
bolehMasuk = rombongan.any(anggota -> anggota.usia > 17);
```

Takeaways

- Dalam desain kelas/objek akan ditemukan banyak sekali kelas/objek yang naturnya bersifat sebagai *collection*
- *Levels of abstraction* di OOP, pada tingkat tinggi biasanya tidak perlu peduli jumlah elemen sebuah *collection* maupun tiap elemen secara individu
  - Kode menjadi lebih *readable*
- Tidak semua bahasa OO menyediakan kelas-kelas *collection* yang "true OO" secara *built-in*
  - Contoh: Collection API di Java tidak "true OO", bisa gunakan library seperti Eclipse Collections

```
boolean anyPeopleHaveCats =
  this.people
    .anySatisfy(person -> person.hasPet(PetType.CAT));

int countPeopleWithCats =
  this.people
    .count(person -> person.hasPet(PetType.CAT));

MutableList<Person> peopleWithCats =
  this.people
    .select(person -> person.hasPet(PetType.CAT));
```

https://www.eclipse.org/collections/